

## PLP - Primer Parcial - 2<sup>do</sup> cuatrimestre de 2014

Este examen se aprueba obteniendo al menos **65 puntos** en total, y al menos **5 puntos** por cada tema. Poner nombre, apellido y número de orden en cada hoja, y numerarlas. Se puede utilizar todo lo definido en las prácticas y todo lo que se dio en clase, colocando referencias claras. El puntaje de los distintos ítems no está necesariamente relacionado con su dificultad.

### Ejercicio 1 - Programación funcional (35 puntos)

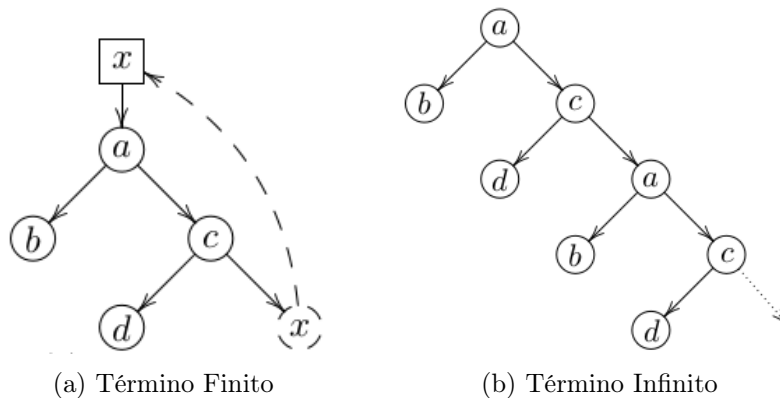
Considerar la siguiente definición de tipo de los árboles regulares (posiblemente) infinitos:

```
data LblTree a b = Nil | Node a (LblTree a b) (LblTree a b) | Lnk b | Lbl b (LblTree a b)
deriving Eq
```

Estos son árboles cuyos subárboles pueden ser etiquetados. Una etiqueta se introduce con un subtérmino de la forma `Lbl x arb1`, es decir, se nombra con  $x$  al subárbol  $arb1$ . Luego, una ocurrencia del subtérmino `Lnk x` en `Lbl x arb1` denota una ocurrencia del subárbol  $arb1$ . Gracias a la utilización de etiquetas, podemos representar con un término finito a un árbol infinito. Por ejemplo, el término

```
arb1 = Lbl 'x' (Node 'a' (Node 'b' Nil Nil) (Node 'c' (Node 'd' Nil Nil) (Lnk 'x')))
```

que se muestra gráficamente en Fig. 1a representa al árbol infinito que se muestra en Fig. 1b.



Notar que el operador `Lbl` actúa como un ligador. Dado `Lbl x arb1`, toda ocurrencia de  $x$  en  $arb1$  es ligada. Como es usual, una ocurrencia es libre si no es ligada.

**Importante:** dar el tipo de cada una de las funciones que defina

- (a) Definir un esquema de recursión estructural del tipo `foldr` para el tipo `LblTree`. Puede utilizar recursión explícita sobre el tipo `LblTree` en su solución.

**Importante:** en los siguientes puntos no puede utilizar recursión explícita sobre ningún tipo.

- (b) Definir la función `varLibres` que dado un árbol etiquetado devuelve una lista con las variables libres. Suponer que no hay apariciones libres de variables ligadas en otra parte del árbol. Por ejemplo, para la siguiente expresión el resultado debe ser `[x, y]`.

```
Lbl 'z' (Node 'a'
  (Node 'b' (Lnk 'y') (Lnk 'x'))
  (Lbl 'y' (Node 'c' (Node 'd' Nil Nil) (Lnk 'x'))))
```

(c) Dada la función `recLT` definida de la siguiente manera:

```
recLT :: c
-> (a -> LblTree a b -> LblTree a b -> c -> c -> c)
-> (b -> c)
-> (b -> LblTree a b -> c -> c)
-> LblTree a b
-> c

recLT cNil fNode fLnk fLbl arb = case arb of
  Nil -> cNil
  Node x hi hd -> fNode x hi hd (recursion hi) (recursion hd)
  Lnk v -> fLnk v
  Lbl v p -> fLbl v p (recursion p)
  where recursion = recLT cNil fNode fLnk fLbl
```

Definir la función *sustituir*, que dada una etiqueta  $x$  y dos árboles etiquetados  $t$  y  $arb$ , genere el árbol etiquetado que se obtiene de sustituir en  $arb$  todos los subtérminos `Lnk x` que corresponden a ocurrencias libres de  $x$  por  $t$ . Notar que la sustitución no tiene efecto cuando las ocurrencias de la variable  $x$  a sustituir ocurren ligadas. Por ejemplo, *sustituir*  $'x'$   $t$  (`Lbl 'x' arb`) = (`Lbl 'x' arb`).

**Ejercicio 2 - Extensión de Cálculo Lambda (40 puntos)** Considere la extensión del cálculo lambda tipado con

- identificadores de funciones recursivas y
- con un nuevo término que nos permite definir funciones recursivas.

Nótese que el conjunto de expresiones de tipos y de valores no sufre modificación alguna.

$$M ::= \dots \mid f \mid \text{letrec } f = M \text{ in } N$$

El símbolo  $f$  representa un identificador de función recursiva arbitrario. Tales identificadores se toman de un conjunto dado  $\mathfrak{F} = \{f_1, f_2, \dots\}$ . El término `letrec  $f$  =  $M$  in  $N$` , asocia a  $f$  la función  $M$  y luego evalúa  $N$ . Tal como ilustra el siguiente ejemplo, tanto  $M$  como  $N$  pueden tener ocurrencias libres de  $f$ :

```
letrec f =  $\lambda x:\text{Nat}.$ if isZero(x) then True else not (f (pred(x))) in f 3
```

Este término reduce al valor `False`.

- Dar una regla de tipado para `letrec`. Nota: tener en cuenta que los contextos  $\Gamma$  ahora no solamente asignan tipos a variables sino que también asignan tipos a identificadores de funciones recursivas.
- Demostrar la validez del siguiente juicio de tipado:

```
{x : Bool}  $\triangleright$  letrec f =  $\lambda y:\text{Bool}.$ succ(f y) in f x : Nat
```

- Dar la semántica operacional de `letrec`. Ayuda: los estados de la “máquina abstracta” determinada por la semántica operacional consisten de pares que contienen términos y un entorno. El entorno es una función que asocia significado a identificadores de funciones recursivas. Por ejemplo,

```
letrec f =  $\lambda x:\text{Nat}.$ if isZero(x) then succ(0) else x * f(pred(x)) in f 0 |  $\emptyset$ 
 $\rightarrow$  f 0 | {f :=  $\lambda x:\text{Nat}.$ if isZero(x) then succ(0) else x * f(pred(x))}
 $\rightarrow$  ( $\lambda x:\text{Nat}.$ if isZero(x) then succ(0) else x * f(pred(x))) 0
      | {f :=  $\lambda x:\text{Nat}.$ if isZero(x) then succ(0) else x * f(pred(x))}
 $\rightarrow^*$  succ(0) | {f :=  $\lambda x:\text{Nat}.$ if isZero(x) then succ(0) else x * f(pred(x))}
```

Aclaración: todas las reglas del cálculo deberán adaptarse a este nuevo entorno. Asumimos que ya fueron modificadas correctamente todas salvo las introducidas por esta nueva extensión.

### Ejercicio 3 - Algoritmo de Inferencia (25 puntos)

Contamos con una extensión del cálculo lambda tipado para soportar árboles binarios alternados, donde todos los elementos en los niveles impares tienen tipo  $\sigma$  y en niveles pares tienen tipo  $\tau$ . Los conjuntos de tipos y términos se extienden de la siguiente manera

$$\begin{aligned}\sigma &::= \text{Arb}_{\sigma,\tau} \\ M &::= \text{Nil}_{\sigma,\tau} \mid \text{Bin}(M, N, O) \mid \text{case } M \text{ of Nil} \rightsquigarrow N, \text{Bin}(i, r, d) \rightsquigarrow O\end{aligned}$$

Las reglas de tipado son las siguientes

$$\begin{array}{c} \frac{}{\emptyset \triangleright \text{Nil}_{\sigma,\tau} : \text{Arb}_{\sigma,\tau}} \quad \frac{\Gamma \triangleright M : \text{Arb}_{\sigma,\tau} \quad \Gamma \triangleright O : \text{Arb}_{\sigma,\tau} \quad \Gamma \triangleright N : \tau}{\Gamma \triangleright \text{Bin}(M, N, O) : \text{Arb}_{\tau,\sigma}} \\[10pt] \frac{\Gamma \triangleright M : \text{Arb}_{\sigma,\tau} \quad \Gamma \triangleright N : \rho \quad \Gamma \cup \{i : \text{Arb}_{\tau,\sigma}, d : \text{Arb}_{\tau,\sigma}, r : \sigma\} \triangleright O : \rho}{\Gamma \triangleright \text{case } M \text{ of Nil} \rightsquigarrow N, \text{Bin}(i, r, d) \rightsquigarrow O : \rho}\end{array}$$

- (a) Extender el algoritmo de inferencia para soportar la extensión descripta.
- (b) Utilizar el algoritmo extendido para tipar o encontrar un error de tipos en la siguiente expresión.

$$(\lambda x. \lambda y. \text{Bin}(x, y, \text{Bin}(x, y, y))) \ 1$$